



Inspecting and optimizing memory usage in Linux

João Marcos Costa
joaomarcos.costa@bootlin.com

© Copyright 2004-2024, Bootlin.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!





- ▶ Embedded Linux engineer and trainer at Bootlin since 2023
 - Embedded Linux experts
 - Engineering services: Linux BSP development, kernel porting and drivers, Yocto/Buildroot integration, real-time, boot-time, security, multimedia
 - Training services: Embedded Linux, Linux kernel drivers, Yocto, Buildroot, graphics stack, boot-time, real-time
- ▶ Open-source contributor
- ▶ Living in **Lyon**, France
- ▶ `joaomarcos.costa@bootlin.com`

<https://bootlin.com/company/staff/joao-marcos-costa/>



Background

- ▶ Project's requirements:
 - use as little memory as possible
 - furthermore, understand how the memory is being used
- ▶ iMX93 Evaluation Kit, with 2 GiB of LPDDR4X RAM
- ▶ a humble enthusiast, not a memory guru!



What is this talk about?

- ▶ Virtual memory
- ▶ Memory used by the kernel vs. by the programs
- ▶ Is it leaking?
- ▶ What if I don't have enough of it?



Introduction



Virtual memory: the basics

- ▶ Physical memory is not directly referred to. Virtual addresses are used instead
- ▶ The address translation is handled by the Memory Management Unit (MMU)
- ▶ The virtual address space is typically divided in 4KiB-long pages
- ▶ Other page sizes do exist (see [Huge pages](#))
- ▶ Those pages are indexed in the MMU's Page Tables



Virtual memory: representing pages

```
include/linux/mm_types.h
```

```
struct page {  
    unsigned long flags; /* Page status, see <include/linux/page-flags.h> */  
    struct list_head lru;  
    struct address_space *mapping;  
    pgoff_t index;  
    atomic_t _refcount; /* Usage count */  
    void *virtual; /* Kernel virtual address */  
};
```

- ▶ This struct represents physical page, not a virtual one
- ▶ Example: 2GiB of RAM \equiv 524288 4KiB-long pages
 - *struct page* is \approx 64-bytes long
 - $524288 \times 64 = 32\text{MiB}$ to represent physical memory pages



Virtual memory: representing pages

```
include/linux/mm_types.h
```

```
struct vm_area_struct {  
    /* VMA covers [vm_start; vm_end) addresses within mm */  
    unsigned long vm_start;  
    unsigned long vm_end;  
    struct mm_struct *vm_mm;    /* The address space we belong to. */  
    const vm_flags_t vm_flags;  
    struct list_head anon_vma_chain;  
    struct anon_vma *anon_vma;  
    /* Function pointers to deal with this struct. */  
    const struct vm_operations_struct *vm_ops;  
    struct file * vm_file;    /* File we map to (can be NULL). */  
    void * vm_private_data;    /* was vm_pte (shared mem) */  
};
```

- ▶ There is not a single struct to represent a virtual page
- ▶ Instead, Linux refers to a range of virtual addresses, or *Virtual Memory Area* (VMA)

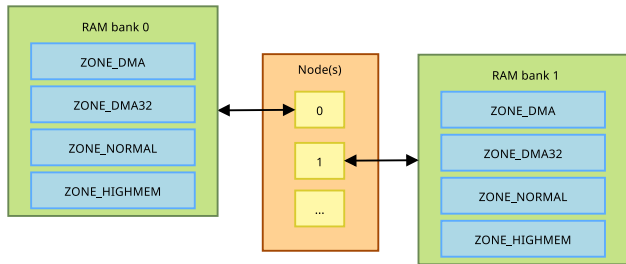


Memory zones

- ▶ Physical memory is divided in zones, rather than being an homogeneous pool of addresses
- ▶ **ZONE_DMA**: the lower 16 MiB, for Direct Memory Access (legacy?)
- ▶ **ZONE_DMA32**: between 16 MiB, and below 4 GiB (64-bit Linux only)
- ▶ **ZONE_NORMAL**:
 - For 32-bit machines, between 16 MiB and 896 MiB
 - For 64-bit machines, all memory above 4 GiB
- ▶ **ZONE_HIGHMEM**: the memory above 896 MiB, only in 32-bit machines



Zones and nodes



- ▶ Zones are attached to nodes, and nodes to CPUs
- ▶ One node per CPU
- ▶ Each node is aware of its zones and their available memory pages
- ▶ Non-Uniform Memory Access (NUMA)



Zones and nodes

512 MiB RAM, 32-bits machine

```
# cat /proc/buddyinfo
Node 0, zone Normal      28      13      8      3      3      1      2      2      2      3      2      51
```

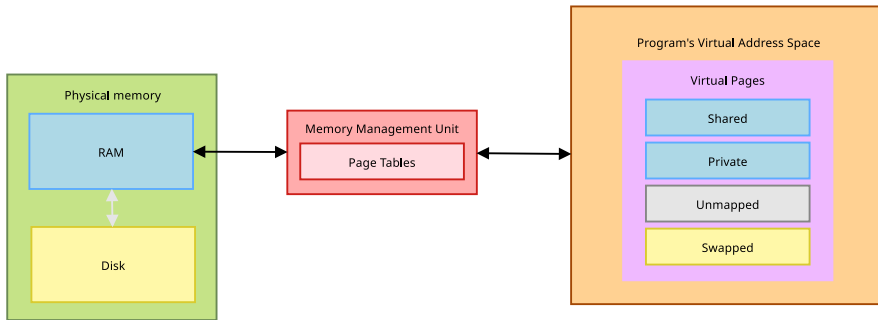
- ▶ Each column represents the number of available consecutive memory chunks of a certain size
- ▶ Every chunk's (i.e. column) size is defined by: $PAGESIZE \times K \times 2^n$
- ▶ e.g.: the first column ($K = 28, n = 0$) stands for 28 chunks of 4096 bytes

32 GiB RAM, 64-bits machine

```
# cat /proc/buddyinfo
Node 0, zone DMA          0          0          0          0          0          0          0          0          1          1          2
Node 0, zone DMA32    2241    1787    1400    1356    935    485    157    39    7    5    0
Node 0, zone Normal  26143  18051  12270  8620  5536  3278  1564  620  206  40  3
```



Virtual memory: what does it provide?



- ▶ Memory is represented in a simpler way: as a uniform and contiguous address space
- ▶ Each process will run in its own isolated address space
- ▶ Primary and secondary memory (i.e., the disk) are abstracted as one
- ▶ Processes can share memory segments (e.g., shared libraries, and IPC)



Memory usage in kernel-space



Memory usage in kernel-space

- ▶ Memory allocations (*vmalloc()* and *kmalloc()*)
- ▶ Modules
- ▶ The kernel's binary itself
- ▶ Low level allocations (simply not tracked)



Examining the boot logs

- ▶ The starting point for our analysis
- ▶ Memory: A/B available (...)
- ▶ B: The total physical memory (minus *OPTEE*)
- ▶ A: B minus the memory reserved by/for the kernel (i.e., reserved field)

Kernel logs from early boot

```
[ 0.000000] Memory: 2012000K/2064384K available (7936K kernel code, 572K rwddata, 1708K rodata, 1280K init, 328K bss, 52384K reserved, 0K cma-reserved)
```



Examining the boot logs

- ▶ `kernel` code: `.text` section of the kernel binary
- ▶ `rwxdata`: initialized (and writable) global and static variables
- ▶ `rodata`: read-only kernel data as constants and strings
- ▶ `init`: initialization code, reclaimed later
- ▶ `bss`: uninitialized data
- ▶ `reserved`: an overall metric for memory reserved by/for the kernel including code, data, and the physical pages (32 MiB, in this case)
- ▶ `cma-reserved`: Contiguous Memory Allocator



Examining the boot logs

mm/mm_init.c

```
static void __init mem_init_print_info(void)
{
    [...]

    pr_info("Memory: %luK/%luK available (%luK kernel code, %luK rwdata, %luK rodata, %luK init, %luK bss, %luK reserved, %luK cma-reserved"
#ifdef CONFIG_HIGHMEM
        ", %luK highmem"
#endif
        ")\n",
        K(nr_free_pages()), K(phypages),
        codesize / SZ_1K, datasize / SZ_1K, rosize / SZ_1K,
        (init_data_size + init_code_size) / SZ_1K, bss_size / SZ_1K,
        K(phypages - totalram_pages() - totalcma_pages),
        K(totalcma_pages)
#ifdef CONFIG_HIGHMEM
        , K(totalhigh_pages())
#endif
    );

    [...]
}
```



Kernel memory usage, but from user-space

Kernel logs from early boot

```
[ 0.000000] Memory: 2012000K/2064384K available (...1280K init...
```

/proc/meminfo

```
root@xxxx-imx93:~# grep MemTotal /proc/meminfo
MemTotal:      2013280 kB
```

- ▶ Total memory slightly increased after `init` segment was retrieved
- ▶ `/proc/meminfo`: significant information about the kernel's memory usage



Examining /proc/meminfo

- ▶ Slab: total memory utilized for caching in-kernel data structures, managed by the Slab allocator
- ▶ KernelStack: memory allocated to kernel stacks
- ▶ PageTables: lowest level arrays of pages for address translation
- ▶ VmallocUsed: the used portion of vmalloc memory area

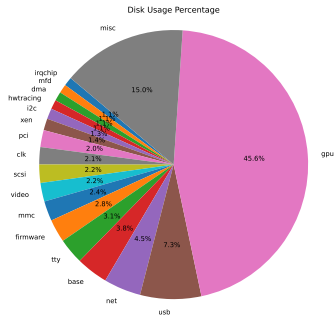
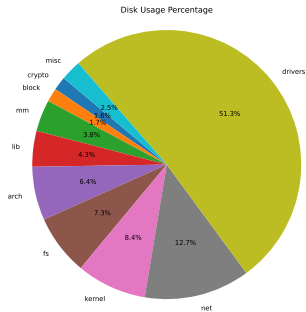
/proc/meminfo

```
# cat /proc/meminfo
```

```
Slab:          9440 kB
KernelStack:   752 kB
PageTables:    628 kB
VmallocUsed:   15204 kB
[...]
```



The kernel binary



- ▶ How much space each component takes in the final binary
- ▶ Rough numbers, based on the disk usage in the build directory
- ▶ The less code, the better!



Memory usage in user-space



First approach: using ps command

```
# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.6  22452  3080 ?        Ss   Jan05    0:17 /sbin/init
root       161  0.0  0.3   5440  1728 ?        Ss   Jan05    0:02 /lib/systemd/systemd-udevd
root       169  0.0  1.1  13120  6024 ?        Ss   Jan05    1:19 /lib/systemd/systemd-journald
root       214  0.0  0.1   2776   844 ?        Ss   Jan05    0:00 /sbin/klogd -n
```

- ▶ Virtual Set Size and Resident Set Size
- ▶ VSZ: total virtual memory size, as listed in `/proc/<PID>/maps`
- ▶ RSS: the memory that is actually mapped to physical pages
- ▶ Equivalent to VIRT and RES in `top` command
- ▶ RSS repeatedly accounts for shared memory areas ⚠



Second approach: using smem command

```
# smem -t
PID User      Command                      Swap    USS    PSS    RSS
290 0          /sbin/getty 38400 tty1       0        80    240   1380
281 0          /sbin/syslogd -n -O /var/lo  0        76    249   1440
284 0          /sbin/klogd -n               0       100    273   1464
  1 0          init [5]                     0       176    286   1172
289 0          /bin/sh /bin/start_getty 11   0        84    310   1576
325 0          smemcap                      0       164    350   1528
292 0          -sh                           0       216    442   1708
273 0          /usr/sbin/dropbear -r /etc/   0       344    455   1340
162 0          /sbin/udev -d                0       792    902   1784
-----
  9 1                               0    2032   3507  13392
```

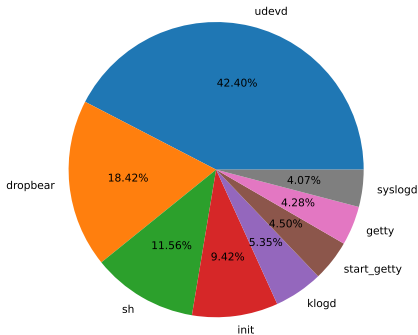
- ▶ Unique Set Size and Proportional Set Size
- ▶ USS: unique (private) memory per process
- ▶ PSS: proportional fraction of a shared memory zone
- ▶ if a 120Kb memory zone is shared among 3 processes, the PSS will be of 40Kb for each
- ▶ more details in `/proc/<PID>/smaps`



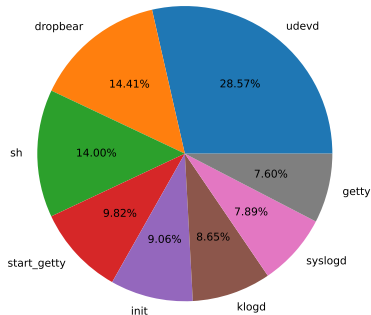
Plotting the memory usage with smem

- ▶ Record the data on the target: `smemcap > smemcap_imx93.tar`
- ▶ Plot it on the host: `smem --pie name -S smemcap_imx93.tar -s [uss|pss]`

Total USS ratio per-process



Total PSS ratio per-process





free command

```
# free -h
```

	total	used	free	shared	buff/cache	available
Mem:	31Gi	18Gi	652Mi	3.5Gi	16Gi	12Gi

- ▶ used: unavailable memory
- ▶ available: used, but can be reclaimed
- ▶ free: not used for anything (wasted?)
- ▶ shared and buff/cache: tmpfs, kernel buffers, page cache, etc.
- ▶ check linuxatemyram.com



Optimization



Smaller code

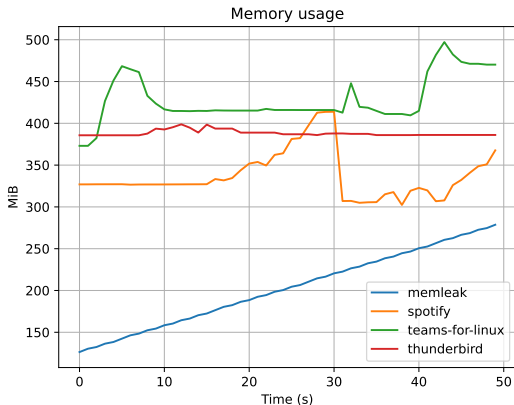
- ▶ Thinking about features in a wider perspective (graphics, networking, filesystems, buses, virtualization, etc) then disabling the corresponding configs
- ▶ Compiler optimization level (size, rather than performance)
- ▶ Disable debugging (`CONFIG_DEBUG_KERNEL`)
- ▶ In iMX93: went from a 32MiB kernel to 11 MiB



Catching memory leaks

▶ Monitoring *procs*

- *free* command, for a global view
- per-process *USS* and/or *RSS*, to find the leak source
- */proc/meminfo*, and */proc/<PID>/smaps* respectively





Catching memory leaks

► Valgrind

- Detects memory management errors: leaks, invalid accesses, bad freeing of heap blocks, etc.
- No need to rebuild your program, but it works better with `-g`

```
#include <stdlib.h>

int main() {
    char *string_a, *string_b;

    string_a = malloc(10); /* 10 bytes leaking */
    free(string_b); /* Invalid free */

    return 0;
}
```



Catching memory leaks

```
valgrind --leak-check=full --show-leak-kinds=all ./memleak
==941479== Memcheck, a memory error detector
==941479== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==941479== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==941479== Command: ./memleak
==941479==
==941479== Conditional jump or move depends on uninitialised value(s)
==941479==    at 0x4845ADF: free (vg_replace_malloc.c:985)
==941479==    by 0x401157: main (memleak.c:7)
==941479==
==941479==
==941479== HEAP SUMMARY:
==941479==    in use at exit: 10 bytes in 1 blocks
==941479==    total heap usage: 1 allocs, 0 frees, 10 bytes allocated
==941479==
==941479== 10 bytes in 1 blocks are definitely lost in loss record 1 of 1
==941479==    at 0x484280F: malloc (vg_replace_malloc.c:442)
==941479==    by 0x401147: main (memleak.c:6)
==941479==
==941479== LEAK SUMMARY:
==941479==    definitely lost: 10 bytes in 1 blocks
==941479==    indirectly lost: 0 bytes in 0 blocks
==941479==    possibly lost: 0 bytes in 0 blocks
==941479==    still reachable: 0 bytes in 0 blocks
==941479==    suppressed: 0 bytes in 0 blocks
==941479==
==941479== Use --track-origins=yes to see where uninitialised values come from
==941479== For lists of detected and suppressed errors, rerun with: -s
==941479== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```



Swap

- ▶ Freeing memory by moving pages to the disk
- ▶ Not quite suited for embedded systems: wearing out flash disks, unresponsiveness (*thrashing*), etc.
- ▶ Can be tuned with `/proc/sys/vm/swappiness`, ranging from 0 to 200
- ▶ Use it to balance swapping and filesystem paging (e.g., 100 means equal IO cost)
- ▶ Setting it to 0...
 - Disables it in a memory control group context
 - Does **not** disable it in a system-wide context

[Documentation/admin-guide/cgroup-v1/memory.rst](#)

Please note that unlike during the global reclaim, limit reclaim enforces that 0 swappiness really prevents from any swapping even if there is a swap storage available. This might lead to memcg OOM killer if there are no file pages to reclaim.



ZRAM: like swap, but better

- ▶ Pages are compressed then stored into a RAM-based block device
- ▶ Faster than disk-based swap

```
Device Drivers --->
[*] Block devices --->
  <M> Compressed RAM block device support
  [*] Write back incompressible or idle page to backing device
  [*] Track zRam block status
```




Out of memory killer

- ▶ Last resource: a process is killed to reclaim memory
- ▶ Heuristic choice: the more memory the process takes, the higher its *badness*

mm/oom_kill.c

```
long oom_badness(struct task_struct *p, unsigned long totalpages)
{
    /* (...) */

    /*
     * The baseline for the badness score is the proportion of RAM that each
     * task's rss, pagetable and swap space use.
     */
    points = get_mm_rss(p->mm) + get_mm_counter(p->mm, MM_SWAPENTS) +
             mm_pgtables_bytes(p->mm) / PAGE_SIZE;

    /* Normalize to oom_score_adj units */
    adj *= totalpages / 1000;
    points += adj;

    return points;
}
```



Out of memory killer

- ▶ Tuning OOM Killer: `/proc/<PID>/oom_score_adj`
- ▶ This adjustment parameter ranges from -1000 to 1000

```
include/uapi/linux/oom.h
```

```
/*  
 * /proc/<pid>/oom_score_adj set to OOM_SCORE_ADJ_MIN disables oom killing for  
 * pid.  
 */  
#define OOM_SCORE_ADJ_MIN    (-1000)  
#define OOM_SCORE_ADJ_MAX    1000
```

- ▶ Example: `echo -1000 > /proc/`pgrep myapp`/oom_score_adj`

```
# cat /proc/`pgrep firefox`/oom_score  
738  
# echo -1000 > /proc/`pgrep firefox`/oom_score_adj  
# cat /proc/`pgrep firefox`/oom_score  
0
```

- ▶ Firefox is now immune to the OOM Killer!

Questions? Suggestions? Comments?

João Marcos Costa
joaomarcos.costa@bootlin.com

Slides under CC-BY-SA 3.0
<https://bootlin.com/pub/conferences/>